

GT4 GridFTP for Developers: The New GridFTP Server

Bill Allcock, ANL
NeSC, Edinburgh, Scotland
Jan 27-28, 2005



Overview

- **Introduction to GridFTP**
- Overview of asynchronous programming
- GridFTP Client Library
- The server Data Storage Interface (DSI)

What is GridFTP?

- A secure, robust, fast, efficient, standards based, widely accepted data transfer protocol
- A Protocol
 - ◆ Multiple independent implementations can interoperate
 - This works. Both the Condor Project at Uwis and Fermi Lab have home grown servers that work with ours.
 - Lots of people have developed clients independent of the Globus Project.
- We also supply a reference implementation:
 - ◆ Server
 - ◆ Client tools (globus-url-copy)
 - ◆ Development Libraries

GridFTP: The Protocol

- FTP protocol is defined by several IETF RFCs
- Start with most commonly used subset
 - ◆ Standard FTP: get/put etc., 3rd-party transfer
- Implement standard but often unused features
 - ◆ GSS binding, extended directory listing, simple restart
- Extend in various ways, while preserving interoperability with existing servers
 - ◆ Striped/parallel data channels, partial file, automatic & manual TCP buffer setting, progress monitoring, extended restart

GridFTP: The Protocol (cont)

- Existing standards
 - ◆ RFC 959: File Transfer Protocol
 - ◆ RFC 2228: FTP Security Extensions
 - ◆ RFC 2389: Feature Negotiation for the File Transfer Protocol
 - ◆ Draft: FTP Extensions
 - ◆ GridFTP: Protocol Extensions to FTP for the Grid
 - Grid Forum Recommendation
 - GFD.20
 - <http://www.ggf.org/documents/GWD-R/GFD-R.020.pdf>



wuftp based GridFTP

Functionality prior to GT3.2

- Security
- Reliability / Restart
- Parallel Streams
- Third Party Transfers
- Manual TCP Buffer Size
- Partial File Transfer
- Large File Support
- Data Channel Caching
- Integrated Instrumentation
- De facto standard on the Grid

New Functionality in 3.2

- Server Improvements
 - Structured File Info
 - MLST, MLSD
 - checksum support
 - chmod support (client)
- globus-url-copy changes
 - File globbing support
 - Recursive dir moves
 - RFC 1738 support
 - Control of restart
 - Control of DC security

New GT4 GridFTP Implementation

- NOT based on wuftp
- 100% Globus code. No licensing issues.
- GT3.9.4 (released in Dec.) has a very solid alpha. It will be in the GT4.0 Final scheduled for 2Q2005.
- wuftp specific functionality, such as virtual domains, will NOT be present
- Has IPV6 support included (EPRT, EPSV), but we have limited environment for testing.
- Based on XIO
- Extremely modular to allow integration with a variety of data sources (files, mass stores, etc.)
- Striping will also be present in 4.0



the globus alliance

www.globus.org

Extensible IO (XIO) system

- Provides a framework that implements a Read/Write/Open/Close Abstraction
- Drivers are written that implement the functionality (file, TCP, UDP, GSI, etc.)
- Different functionality is achieved by building protocol stacks
- GridFTP drivers will allow 3rd party applications to easily access files stored under a GridFTP server
- Other drivers could be written to allow access to other data stores.
- Changing drivers requires minimal change to the application code.



Striped Server

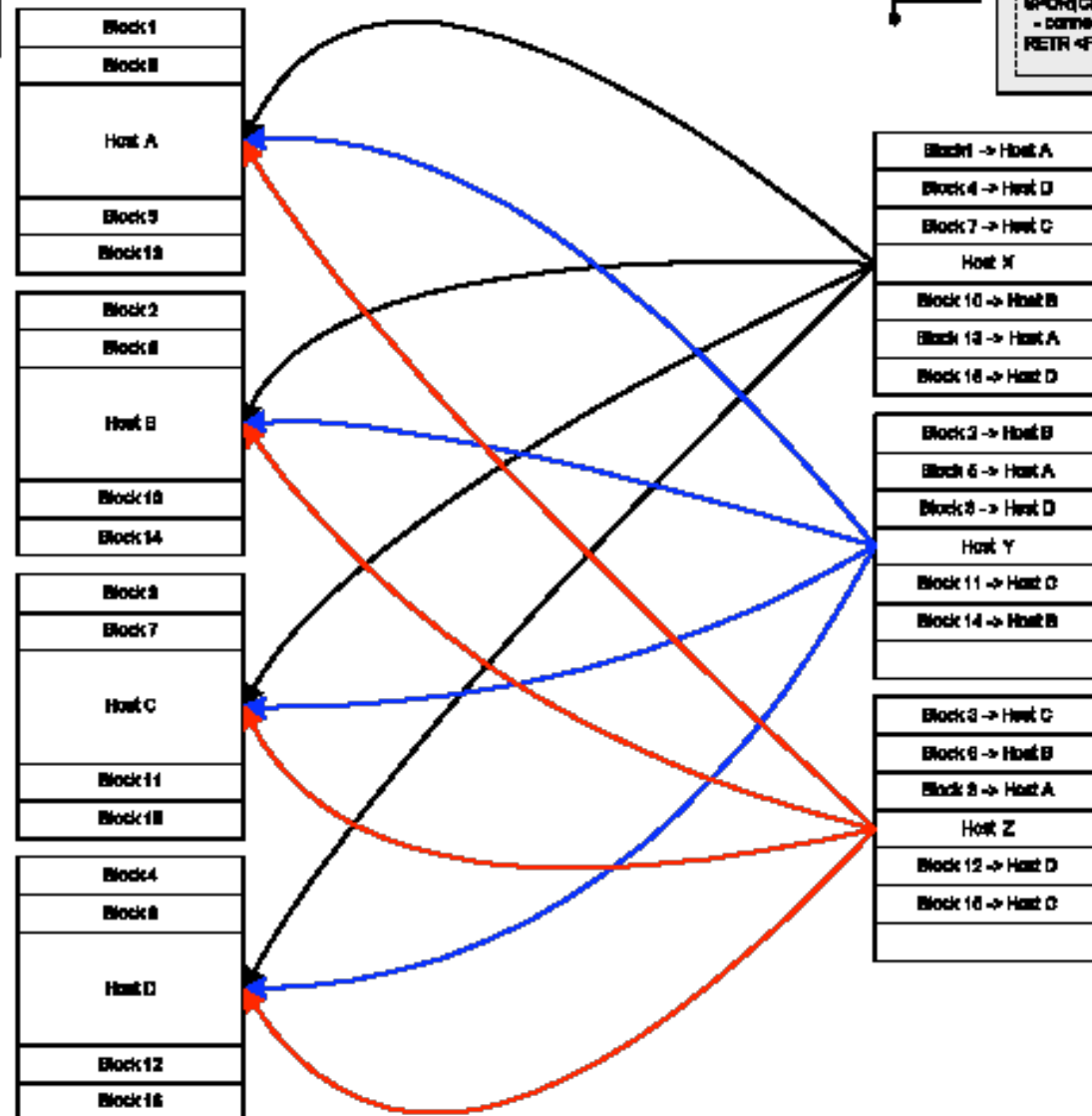
- Multiple nodes work together and act as a single GridFTP server
- An underlying parallel file system allows all nodes to see the same file system and must deliver good performance (usually the limiting factor in transfer speed)
 - ◆ I.e., NFS does not cut it
- Each node then moves (reads or writes) only the pieces of the file that it is responsible for.
- This allows multiple levels of parallelism, CPU, bus, NIC, disk, etc.
 - ◆ Critical if you want to achieve better than 1 Gbs without breaking the bank

16-Nov-03

GridFTP Striped Transfer

MODE P
 SPPE (Listen)
 - returns list of host:port pairs
 BTCLR <File Name>

MODE P
 SPOR (Connect)
 - connect to the host:port pairs
 RETR <File Name>



GridFTP: Caveats

- Protocol requires that the sending side do the TCP connect (possible Firewall issues)
- Client / Server
 - ◆ Currently, no simple encapsulation of the server side functionality (need to know protocol), therefore Peer to Peer type apps VERY difficult
 - A library with this encapsulation is on our radar, but no timeframe.
 - ◆ Generally needs a pre-installed server
 - Looking at a “dynamically installable” server

Overview

- Introduction to GridFTP
- **Overview of asynchronous programming**
- GridFTP Client Library
- The server Data Storage Interface (DSI)

Asynchronous Programming

- There are 3 basic event models
 - ◆ Blocking: Code does not make progress until event handling is finished.
 - ◆ Non-blocking: Code can make progress, but there is typically a large case or if structure.
 - ◆ Asynchronous: No in-line path of execution. Event handlers are registered and executed as needed.

Asynch Programming is complicated

- There is no in-line logic that can be easily looked at and understood.
- All state needs to be packaged up in a structure and passed through.
- You need to be careful of race conditions.
- The event handling system is not really “visible” so it seems like there is some “magic” involved.

The callback is everything

- The term callback may be a bit confusing, because it does not necessarily “call back” to some other process.
- Think of it as “Now that I am done, what should happen next?”

```
main()  
{  
    1();  
    2();  
    3();  
}  
  
2(cb=3) { ... return();}  
3(cb=done) {... return();}  
main()  
{  
    1(cb =2);  
}
```



Example Code

In main():

```
bytes_read = fread(buffer, 1, MAX_BUFFER_SIZE, fd);
globus_ftp_client_register_write(&handle, buffer, bytes_read, global_offset,
                                feof(fd), data_cb, (void *) fd);
```

In data_cb():

```
    if(!feof(fd)
    {
        bytes_read = fread(buffer, 1, MAX_BUFFER_SIZE, fd);
        if (ferror(fd))
        {
            printf("Read error in function data_cb; errno = %d\n", errno);
            globus_mutex_unlock(&lock);
            return;
        }
        globus_ftp_client_register_write(
            handle,
            buffer,
            bytes_read,
            global_offset,
            feof(fd),
            data_cb,
            (void *) fd);
        cb_ref_count++;
        global_offset += bytes_read;
    }
```

Globus Thread Abstraction

- With Globus libraries, you write threaded and non-threaded code the same way.
- use `globus_cond_wait` and `globus_cond_signal`
 - ◆ in a threaded build they translate to the standard pthread calls
 - ◆ in a non-threaded they translate to `globus_poll_blocking` and `globus_signal_poll`
- This allows the same code to be built either threaded or non-threaded.



Non-Threaded

- During initialization the XIO select poller callback is registered in the callback library queue. It is always ready.
- Registering your callback places it in the same queue.
- `globus_cond_wait` calls `globus_poll_blocking` which initiates the callback library queue processing. This will not return (in general) until `globus_cond_signal` (`globus_signal_poll`) is called.
- Callbacks can be ready immediately or after a wait time, they can be one-shot or periodic.
- If nothing else is ready, XIO select poller determines how long before the next callback will be ready and sleeps till then.
- So callbacks get queued and executed from either the callback library or XIO select poller.

Threaded

- In this case, things work as expected ☺
- `globus_cond_wait` calls `pthread_cond_wait` and puts the main thread to sleep.
- The select loop runs in its own thread.
- `globus_cond_signal` calls `pthread_cond_signal` and wakes up the thread waiting on the cond (typically main).
 - ◆ Note that POSIX allows the thread to wake up arbitrarily and so the `cond_wait` should be enclosed in some sort of while (!done) loop

Lets look at the web examples

- <http://www-unix.globus.org/toolkit/docs/3.2/developer/globus-async.html>
- `wget http://www-unix.mcs.anl.gov/~allcock/dev-ex.tar`
- `globus-makefile-header -flavor=gcc32dbg
globus_common > makefile_header`

Overview

- Introduction to GridFTP
- Overview of asynchronous programming
- **GridFTP Client Library**
- The server Data Storage Interface (DSI)

Writing a GridFTP Client

- Module Activation / Initialization
- Check Features
- Select Mode
- Set Attributes
- Enable any needed plug-ins
- Execute the operation
- Module Deactivation / Clean up

Initialization

- `globus_module_activate(GLOBUS_FTP_CLIENT_MODULE)`
- Must be called in any program that use the client library.
- Will automatically call `module_activate` for any required lower level modules (like `globus_io`)

Checking Features

- call `globus_ftp_client_features_init`
- then call `globus_ftp_client_feat`
 - ◆ this is a non-blocking call, so you will need to wait on it to finish.
 - ◆ you need only call this once
- Once `globus_ftp_client_feat` has returned, `globus_ftp_client_is_feature_supported` can be called as often as necessary for the various features.

Attributes

- Very powerful feature and control much of the functionality
- Two types of attributes:
 - ◆ Handle Attributes: Good for an entire session and independent of any specific Operation
 - ◆ Operation Attributes: Good for a single operation.
- Files:
 - ◆ globus_ftp_client_attr.c
 - ◆ globus_i_ftp_client.h

Attributes (Cont)

- Handle Attributes:
 - ◆ Initialize/Destroy/Copy Attribute Handle
 - ◆ Connection Caching: Either all, or URL by URL.
 - ◆ Plugin Management: Add/Remove Plugins

Attributes (Cont)

- Operation Attributes

- ◆ Parallelism
- ◆ Striped Data Movement
- ◆ Striped File Layout
- ◆ TCP Buffer Control
- ◆ File Type
- ◆ Transfer Mode
- ◆ Authorization/Privacy/Protection

- Functions

- ◆ `globus_ftp_client_operationattr_set_<attribute>(&attr, &<attribute_struct>)`
- ◆ `globus_ftp_client_operationattr_get_<attribute>(&attr, &<attribute_struct>)`

Attributes (Cont)

- Example Code (structs and enums in globus_ftp_control.h):

```
globus_ftp_client_handle_t      handle;
globus_ftp_client_operationattr_t attr;
globus_ftp_client_handleattr_t  handle_attr;
globus_size_t                   parallelism_level = 4;
globus_ftp_control_parallelism_t parallelism;
globus_ftp_control_layout_t     layout;

globus_module_activate(GLOBUS_FTP_CLIENT_MODULE);
globus_ftp_client_handleattr_init(&handle_attr);
globus_ftp_client_operationattr_init(&attr);
parallelism.mode = GLOBUS_FTP_CONTROL_PARALLELISM_FIXED;
parallelism.fixed.size = parallelism_level;
globus_ftp_client_operationattr_set_mode(&attr,
    GLOBUS_FTP_CONTROL_MODE_EXTENDED_BLOCK);
globus_ftp_client_operationattr_set_parallelism(&attr, &parallelism);
globus_ftp_client_handle_init(&handle, &handle_attr);
```

Mode S versus Mode E

- Mode S is stream mode as defined by RFC 959.
 - ◆ No advanced features accept simple restart
- Mode E enables advanced functionality
 - ◆ Adds 64 bit offset and length fields to the header.
 - ◆ This allows discontinuous, out-of-order transmission and along with the SPAS and SPOR commands, enable parallelism and striping.
- Command:

```
globus_ftp_client_operationattr_set_mode(&attr, GLOBUS_FTP_CONTROL_MODE_EXTENDED_BLOCK);
```

Plug-Ins

- Interface to one or more plug-ins:
 - ◆ Callouts for all interesting protocol events
 - Allows monitoring of performance and failure
 - ◆ Callins to restart a transfer
 - Can build custom restart logic
- Included plug-ins:
 - ◆ Debug: Writes event log
 - ◆ Restart: Parameterized automatic restart
 - Retry N times, with a certain delay between each try
 - Give up after some amount of time
 - ◆ Performance: Real time performance data

Plug-Ins (Cont.)

- Coding:

- ◆ `globus_ftp_client_plugin_t *plugin;`
- ◆ `globus_ftp_client_plugin_set_<type>_func`
 - Macro to make loading the struct easier
- ◆ `globus_ftp_client_handleattr_add_plugin(attr, plugin)`

- Files:

- ◆ `globus_ftp_client_plugin.h`
- ◆ `globus_ftp_client.h`
- ◆ `globus_ftp_client_plugin.c`
- ◆ Also some internal .h files



Plug-Ins (Cont.)

- A plugin is created by defining a `globus_ftp_client_plugin_t` which contains the function pointers and plugin-specific data needed for the plugin's operation. It is recommended that a plugin define a `globus_module_descriptor_t` and plugin initialization functions, to ensure that the plugin is properly initialized.
- Every plugin must define copy and destroy functions. The copy function is called when the plugin is added to an attribute set or a handle is initialized with an attribute set containing the plugin. The destroy function is called when the handle or attribute set is destroyed.

Plug-Ins (Cont.)

- Essentially filling in a structure of function pointers:
 - ◆ Operations (Put, Get, Mkdir, etc)
 - ◆ Events (command, response, fault, etc)
- Called only if both the operation and event have functions defined
- Filtered based on `command_mask`

High Level Calls

- globus_ftp_client_put/get/3rd Party

- Function signature:

```
globus_result_t globus_ftp_client_get  
(globus_ftp_client_handle_t *handle,  
  const char *url,  
  globus_ftp_client_operationattr_t *attr,  
  globus_ftp_client_restart_marker_t *restart,  
  globus_ftp_client_complete_callback_t  
  complete_callback,  
  void *callback_arg)
```

Example: globus_ftp_client_put_test.c

Parallel Put/Get

- Parallelism is hidden. You are not required to do anything other than set the attributes, though you may want to for performance reasons.
- Doc needs to be updated. Does not have enums or structures. Look in `globus_ftp_control.h`

Deactivate / Cleanup

- Free any memory that *you* allocated
- Call the necessary destroy and deactivate functions:

```
globus_ftp_client_handleattr_destroy(&handle_attr);  
globus_ftp_client_operationattr_destroy(&operation_attr);  
globus_ftp_client_handle_destroy(&handle);  
globus_module_deactivate(GLOBUS_FTP_CLIENT_MODULE);
```

- <http://www.globus.org/developer/api-reference.html>

Overview

- Introduction to GridFTP
- Overview of asynchronous programming
- GridFTP Client Library
- **The server Data Storage Interface (DSI)**

New Server Architecture

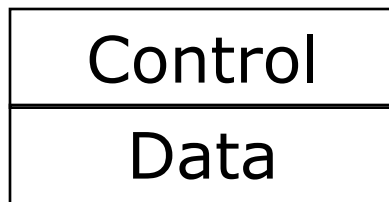
- GridFTP (and normal FTP) use (at least) two separate socket connections:
 - ◆ A control channel for carrying the commands and responses
 - ◆ A Data Channel for actually moving the data
- Control Channel and Data Channel can be (optionally) completely separate processes.
- A single Control Channel can have multiple data channels behind it.
 - ◆ This is how a striped server works.
 - ◆ In the future we would like to have a load balancing proxy server work with this.

New Server Architecture

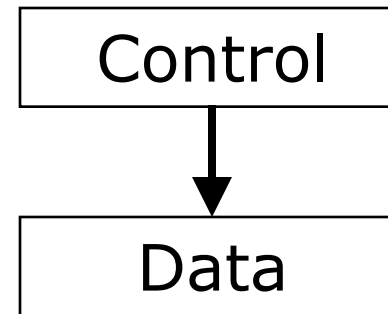
- Data Transport Process (Data Channel) is architecturally, 3 distinct pieces:
 - ◆ The protocol handler. This part talks to the network and understands the data channel protocol
 - ◆ The Data Storage Interface (DSI). A well defined API that may be re-implemented to access things other than POSIX filesystems
 - ◆ ERET/ESTO processing. Ability to manipulate the data prior to transmission.
 - currently handled via the DSI
 - In V4.2 we to support XIO drivers as modules and chaining
- Working with several groups to on custom DSIs
 - ◆ LANL / IBM for HPSS
 - ◆ UWis / Condor for NeST
 - ◆ SDSC for SRB

Possible Configurations

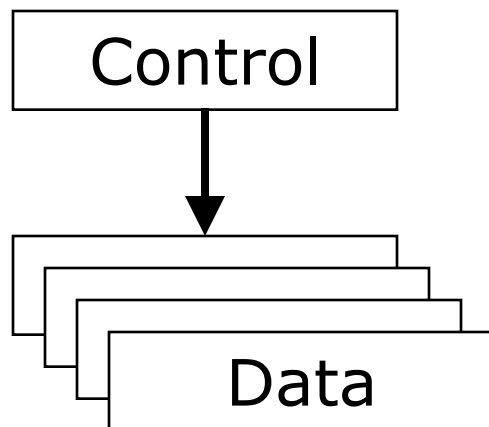
Typical Installation



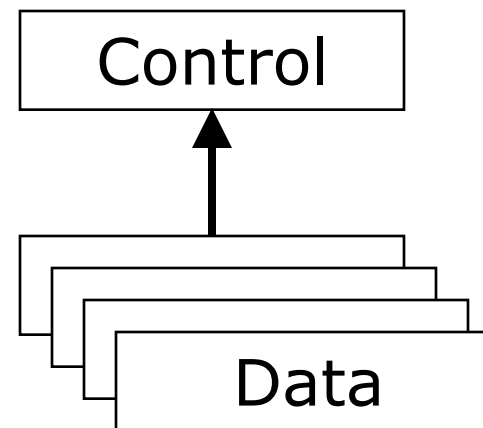
Separate Processes



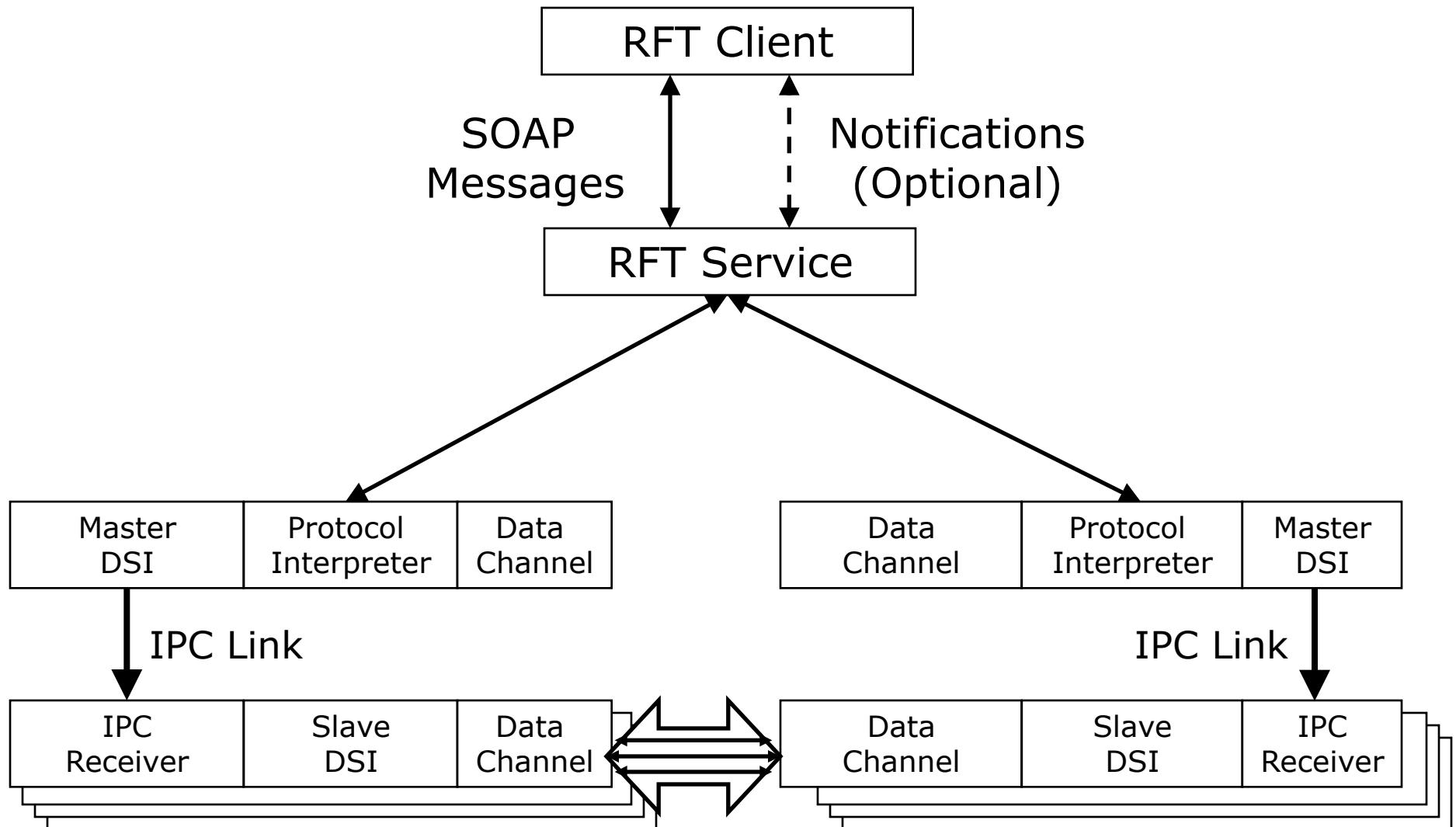
Striped Server



Striped Server (future)



Third Party Transfer



The Data Storage Interface (DSI)

- Unoriginally enough, it provides an interface to data storage systems.
- Typically, this data storage system is a file system accessible via the standard POSIX API, and we provide a driver for that purpose.
- However, there are many other storage systems that it might be useful to access data from, for instance HPSS, SRB, a database, non-standard file systems, etc..



The Data Storage Interface (DSI)

- Conceptually, the DSI is very simple.
- There are a few required functions (init, destroy)
- Most of the interface is optional, and you can only implement what is needed for your particular application.
- There are a set of API functions provided that allow the DSI to interact with the server itself.
- Note that the DSI could be given significant functionality, such as caching, proxy, backend allocation, etc..



Developer Implemented Functions

- Below is the structure used to hold the pointers to your functions.
- This can be found in <install>/source-trees/gridftp/server/src

```
typedef struct globus_gfs_storage_iface_s
{
    int descriptor;

    /* data conn funcs */
    globus_gfs_storage_data_t active_func;
    globus_gfs_storage_data_t passive_func;
    globus_gfs_storage_data_destroy_t data_destroy_func;

    /* session initiating functions */
    globus_gfs_storage_init_t init_func;
    globus_gfs_storage_destroy_t destroy_func;
    globus_gfs_storage_command_t command_func;
    globus_gfs_storage_stat_t stat_func;

    /* transfer functions */
    globus_gfs_storage_transfer_t list_func;
    globus_gfs_storage_transfer_t send_func;
    globus_gfs_storage_transfer_t recv_func;
    globus_gfs_storage_trev_t trev_func;

    globus_gfs_storage_set_cred_t set_cred_func;
    globus_gfs_storage_buffer_send_t buffer_send_func;
} globus_gfs_storage_iface_t;
```

Master vs. Slave DSI

- If you wish to support striping, you will need two DSIs
- The Master DSI will be in the PI or front end. It must implement all functions (that you want to support).
 - ◆ Usually, this is relatively trivial and involves minor processing and then “passing” the command over the IPC channel to the slave DSI
- Any functions not implemented will be handled by the server if possible (non-filesystem, active, list)
- All DSI's must implement the `init_func` and `destroy_func` functions.

Slave Functions

- The slave DSI does the real work. It typically implements the following functions:
 - ◆ send_func: This function is used to send data from the DSI to the server (get or RETR)
 - ◆ recv_func: This function is used to receive data from the server (put or STOR)
 - ◆ stat_func: This function performs a unix stat, i.e. it returns file info. Used by the list function
 - ◆ command_func: This function handles simple (succeed/fail or single line response) file system operations such as mkdir, site chmod, etc.

Slave Functions (cont)

- If you implement active/passive (you normally shouldn't) you will need to implement `data_destroy` to free the data channel memory.
- The `set_cred` function normally does not need to be implemented.

Additional Master Functions

- As noted before, the master should (must?) implement all functions. Besides the sender functions, these include:
 - ◆ `active_func`: This is for when the DSI will be doing a TCP connect.
 - The master figures out who gets what IP/port info and then passes it through.
 - The slave should not need to implement this. The server can handle this for you.
 - ◆ `passive_func`: The counter-part to the `active_func` when the DSI will be the listener
 - ◆ `list_func`: This should be passed through and will handle LIST, NLST, MLST, etc..

Additional Master Functions

- There are also some utility functions the master should (must?) implement:
 - ◆ `data_destroy_func`: Frees the memory associated with the data channel. This should be a simple pass through, unless you implement your own active/passive functions.
 - ◆ `trev_func`: This handles the restart and performance markers, but should be a simple pass through
- If you choose not to implement any of these functions you need to have a good reason.



IPC Calls

- These calls are how the master DSI “passes” the call to the slave DSI
- The IPC calls are basically the same as the DSI calls.
 - ◆ globus_gfs_ipc_iface_stat_t stat_func;
 - ◆ globus_gfs_storage_stat_t stat_func;
- These calls implement an internal, binary protocol to transfer the necessary structures between the front end and the back end.
- The IPC receiver receives the message and then invokes the sender DSI. The sender DSI does not know, nor does it need to know, whether it is local or remote.



Helper Functions that should be used

- When implementing the DSI functions, the following helper functions should be called:
 - ◆ `<function>_finished`: This tells the server that a specific function (such as `recv`) has completed
 - all functions have finished functions. There is also a generic `finished`. The `send` and `recv` also have `start` calls.
 - ◆ `register[read|write]`: This is how file data is transferred between the DSI and the server.
 - ◆ `bytes_written`: This should be called anytime the DSI successfully completes a write to its own storage system. This allows performance and restart markers to be generated

Helper Functions that should be used

- ◆ `get_concurrency`: Tells you the number of outstanding reads or writes you should have based on the parallelism.
- ◆ `get_blocksize`: This indicates the buffer size that you should exchange with the server via the `register_[read|write]`.
- ◆ `get_[read|write]_range`: This tells the DSI which data it should be sending.
 - This handles striping (this DSI only needs to send a portion of the file), restart (including “holey” transfers), and partial files.
 - `read` should be called repeatedly until it returns zero.
 - `write` is only a hint (you have to write where the offset tells you) and should only be called once.